Paper 133-2011

# Romancing Your Data: The Getting-to-Know-You Phase

Carole Jesse, Prime Therapeutics, Eagan, MN, USA

## ABSTRACT

The ability to gather and store large amounts of data has become commonplace. As these capabilities have grown, so has database complexity. The deluge of corporate acquisitions further complicates things, driving more undocumented data sources into the hands of new database users, often without the institutional knowledge to go with them. Thus, analysts must develop the ability to discover the complexity of a database on their own.

This paper focuses on Oracle® databases and use of SAS/ACCESS® Interface to ORACLE to explore the Oracle Data Dictionary Views that contain the database metadata. We focus on metadata summaries of interest to a user new to the database. The metadata is presented through a series of programs that use Base SAS®, specifically the SQL Procedure, macro language, and the Output Delivery System (ODS).

The target audience for this paper includes anyone who uses Oracle databases to solve business problems. Some basic understanding of Base SAS PROC SQL, macro language, and the SAS/ACCESS Interface to ORACLE product is helpful, but not required.

## INTRODUCTION

The users of an Oracle database have access to the tables and views within one or more schemas in the database. Statistical Modelers and Business Analysts are the typical users of the data, and the term 'analyst' will be used here broadly for 'anyone who uses an Oracle database to solve business problems.' An analyst's main interest in the database is to solve business problems that have data-driven solutions.

An analyst may be lucky enough to receive a mapping that includes tables, variable names, relationships, and definitions based on business or institutional knowledge. The mapping might even be a searchable web-based tool. Although a mapping like this is useful, it might not include all the information needed to construct efficient queries to the database. Alternatively, the analyst might have access to a coworker or boss that has practical understanding of the database AND the time to answer their questions.

What if the analyst is not so fortunate? The good news is that every Oracle database includes many useful views containing the database 'metadata'. Just to be clear, the definition of 'metadata' is 'data about data.' To illuminate the 'data about data' concept, consider what happens when you take a picture on a digital camera. The image is stored in a compression format such as a JPEG (Joint Photographic Experts Group). This is the 'file' you open to view the image, and the image is the data. There are also metadata stored with the JPEG containing information about the image. This digital photo metadata includes things like the camera make and model, exposure settings, creation date, and image resolution. The JPEG file's metadata is revealed when working with the image in a photo editing software.

The metadata associated with an Oracle database is similar to this digital photography example. The database metadata are stored within views in the Oracle Data Dictionary, part of the database architecture, and available to all users of the database. You just need to know where these views are, which are most useful to an analyst (as opposed to a Database Administrator), and how to query them.

This paper focuses on the use of SAS/ACCESS Interface to ORACLE to access and explore the Oracle Data Dictionary Views, with emphasis on aspects of interest to a new database user. While there are many Oracle Data Dictionary Views, the 80/20 rule applies: the bulk of the useful information comes from just a handful of the views. Once introduced to these few, the reader will be more comfortable exploring others.

It is worth noting that while this paper focuses on Oracle as the Relational Database Management System (RDBMS), the concepts are easily extended to other RDBMS, and the code can be modified to accommodate any of the other SAS/Access Interface products for RDBMS. Other popular RDBMS include Teradata®, Netezza®, and MS SQL Server®, for which there are SAS/Access Interface products.

The Oracle SYS Views are presented in this paper in a series of SAS programs, each with a specific purpose. All code, and the resulting output, is developed using SAS Version 9.1.3 for the UNIX environment, and a SAS Enterprise Guide Version 4.1 interface. In addition, the version of Oracle referenced for this work is 10g. Readers might encounter differences when running the code, depending on local configuration.

## A GENTLE OVERVIEW OF ORACLE® DATABASE ARCHITECTURE

The architecture of an Oracle database is made up of one or more schemas. A schema is a collection of tables and/or views. Most of the schemas contain the Business Data of interest to the analyst, i.e. the non-metadata data. Every Oracle database also has a schema called SYS. The views in the SYS schema are where the bulk of the user-accessible metadata is stored. The naming convention of the SYS Views is standardized, and this discussion is

applicable irrespective of the Oracle 10g installation. Of particular interest are the SYS Views whose names begin with ALL_, USER_, and DBA_. Figure 1 below illustrates the typical architecture of an Oracle database.

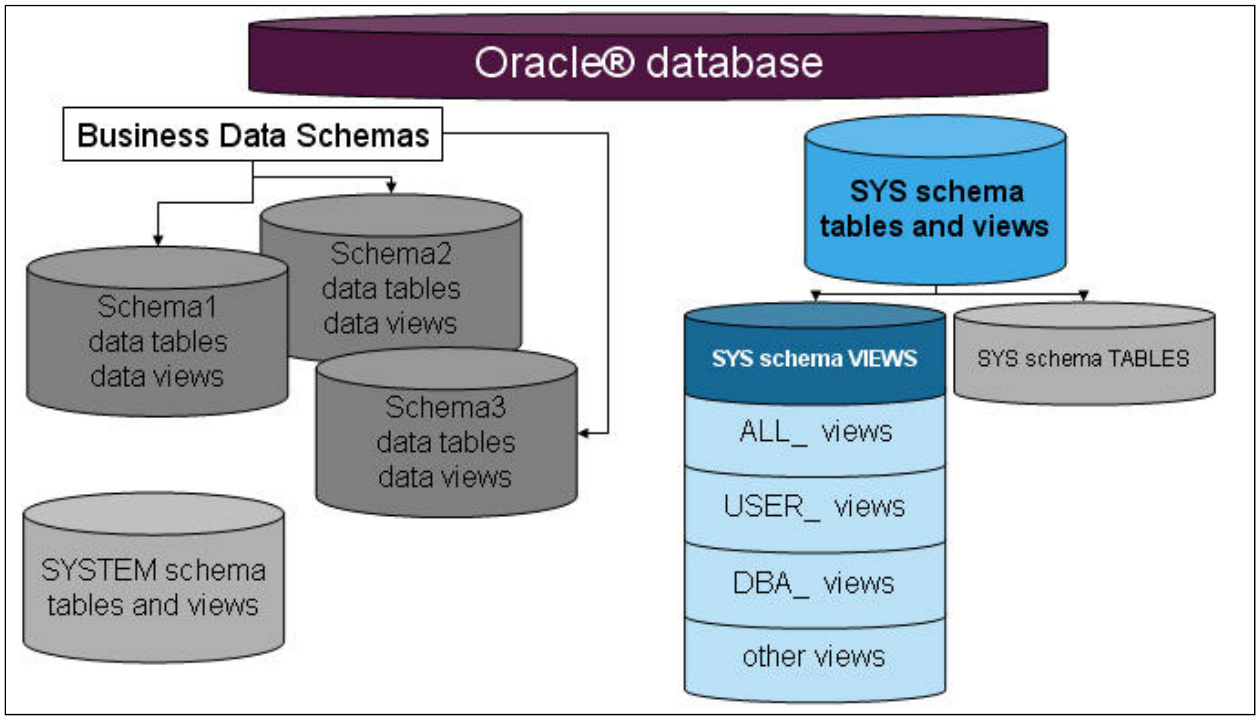


**Figure 1. Oracle® Database Architecture**

As illustrated in Figure 1, the separation of the SYS Views emphasizes three main families: ALL_, USER_, and DBA_. The views in the ALL_ family have names that start with 'ALL_'. The views in the USER_ family have names that start with 'USER_' and the views in the DBA_ family have names that start with 'DBA_'. This taxonomy makes it very easy to separate the views across the three families for exploration.

Since most analysts are not Database Administrators, I do not dwell on the DBA_ views, which require special permissions. Rather, I focus on a handful of the ALL_ and USER_ views.

## A GENTLE OVERVIEW OF SAS/ACCESS® INTERFACE TO ORACLE QUERIES

Before diving into the view queries, a review of SAS syntax is in order. The SAS/ACCESS Interface to ORACLE product allows two types of queries in the SQL Procedure: 'LIBNAME queries' and 'Pass-Through Facility queries.'

### LIBNAME QUERY

A LIBNAME query first assigns a libref directly to a schema in the Oracle database via The LIBNAME Statement. Subsequent to the LIBNAME statement, you can code your PROC SQL query just as if you are working with a SAS data set.

Here is the generic syntax of a LIBNAME statement to an Oracle schema, with Figure 2 showing an example of its use:

```
LIBNAME  libref  oracle  USER='ORACLE-user-name'  PASSWORD='ORACLE-password'
         PATH="ORACLE-database-specification"  SCHEMA=schema-name;
```

```
LIBNAME  db1_sch1  oracle  USER='cjesse'  PASSWORD='bWFLA$01e'
        PATH="db1.server1.com"  SCHEMA=sch1;

/* LIBNAME query to table TBL1 in the SCH1 schema of the DB1 ORACLE database */
PROC SQL;
 CREATE table WORK.TBL1 as select col1, col2, col3 from db1_sch1.TBL1;
QUIT;

/* Alternatively, a DATA step. */
DATA WORK.TBL1;
  SET db1_sch1.TBL1;
  KEEP col1 col2 col3;
RUN;
```

**Figure 2. Example: LIBNAME query in SAS/ACCESS Interface to ORACLE**

## PASS-THROUGH FACILITY QUERY

Pass-through SQL requires The CONNECT Statement, within PROC SQL, which establishes a connection to the database using the user's database credentials. There is no need for a LIBNAME statement.

Here is the generic syntax of a CONNECT statement, within PROC SQL, to an Oracle database. Figure 3 provides an example of its use.

```
CONNECT TO oracle <AS Alias>
        (USER='ORACLE-user-name'  PASSWORD='ORACLE-password'
         PATH="ORACLE-database-specification");
```

```
PROC SQL;
CONNECT TO oracle AS db1 (USER='cjesse'  PASSWORD='bWFLA$01e'
                          PATH="db1.server1.com");
  CREATE table WORK.TBL1 as
  SELECT *
  FROM connection to db1
  (SELECT col1, col2, col3 from sch1.tbl1);
  DISCONNECT FROM db1;
QUIT;
```

**Figure 3. Example: Pass-Through Facility query in SAS/ACCESS Interface to ORACLE**

It is important to be familiar with both the LIBNAME query and Pass-Through Facility query. The LIBNAME approach leads to simpler code, and, in many cases, leads to a sufficient query. However, the Pass-Through Facility approach will trace Oracle errors and display them in the log, which the LIBNAME approach will not. Thus, Pass-Through Facility queries are good for troubleshooting connection issues. Another advantage of Pass-Through Facility queries is the flexibility to combine SAS functions/formats and SQL functions/formats with two select statements. In addition, and as we will see later in this paper, Pass-Through Facility queries may actually be required for working effectively with certain Oracle database objects (i.e., tables that are partitioned).

## MACRO VARIABLE NAMING CONVENTIONS

All of the SAS code presented here is ripe for re-use across multiple databases, so all the programs are 'macro-ized' appropriately to reduce the need for code edits. Specifically six macro variables crop up in the SAS code. All programs require three of the macro variables. The other three macro variables are used as needed in each program. Below are the definitions and syntax requirements for these six macro variables. Examples of the macro variable assignments used for the code demonstration are shown in Figure 4.

### PRIMARY MACRO VARIABLES, DATABASE CONNECTION DETAILS

**&ODBcred :** The user's credentials to the Oracle database. Credentials are Case Sensitive.
General Syntax: `%let ODBcred= user='ORACLE-user-name'  password='ORACLE-password';`

**&ODBlong :** The path specification to the Oracle database.
General Syntax: `%let ODBlong=ORACLE-database-specification;`

**&ODBshrt :** The alias to the Oracle database in the CONNECT statement.
General Syntax: `%let ODBshrt=ORACLE-database-alias;`

### SECONDARY MACRO VARIABLES

**&unixpath :** The UNIX path for permanently saved output files for the ODS HTML statements. This is needed only if you choose to create the HTML output.
General syntax: `%let unixpath= /YourUNIXPath/;`

**&OWNlong :** A particular Schema Name. Case sensitive, ALL CAPS, in single quotes.
General Syntax: `%let OWNlong='SCHEMA-NAME';`

**&TBL :** A particular table or view name within a schema. Case sensitive, ALL CAPS, in single quotes.
General Syntax: `%let TBL='TABLE-NAME';`

```
/* Macro Variable Assignments used to demonstrate SAS programs and create output */
/* (the USER INPUTS to each program) */

/* Begin USER INPUTS */
%let ODBcred=  user='cjesse' pw='bWFLA$01e'; /* credentials to the DB*/
%let ODBlong=  database1.server2.com;        /* path to the DB */
%let ODBshrt=  db1sv2;                       /* alias for the DB */
%let unixpath= /userv1/grp1/cjesse/;     /* UNIX path for output, Case Sensitive */
%let OWNlong=  'OWNER5';                 /* Schema name, ALL CAPS, single quotes */
%let TBL=      'TABLE236';               /* Table name, ALL CAPS, single quotes */
/* End USER INPUTS */
```

**Figure 4. Example: Macro Variable Assignments**

## USING SYS.ALL_VIEWS FOR INITIAL EXPLORATION OF THE SYS VIEWS

With a working knowledge of Oracle database architecture, PROC SQL query types, and the macro variable naming conventions, we can begin the exploration of the SYS Views. When I began my adventure with the Oracle Data Dictionary, I started by looking at the details of all the SYS Views. I focused on the ALL_ and USER_ families since these are accessible to analysts.

The code presented in this section has two parts for initial exploration of the SYS Views. The first part creates a listing of view names within the SYS schema coming from any of the three SYS View families ALL_, USER_, DBA_. As mentioned previously, the most useful metadata for an analyst are in a handful of views within ALL_ and USER_, but inclusion of DBA_ might be helpful for further examination.

The second part of the code demonstrates a macro for drilling down further into the contents of particular views. I find it handy to see the variable names within a view, but also to see the first few values for each variable. This is a great aid in gaining understanding about what information is contained in a particular view, and helps identify the variables of most interest. The macro creates an output listing for each view contained in a list of view names you choose.

If you use SAS Enterprise Guide as an interface, the results generated by these programs are viewable in HTML for convenient browsing. If you are running interactive SAS on the UNIX side, you might find it useful to utilize ODS to HTML to create permanent files for viewing the results in a web browser.

Recently I had access to ten Oracle databases. When running this code for each of the ten databases there were, on average, roughly 286 views within both the ALL_ and USER_ families. Some variation existed in the number of views present from one database to another. However, there was a core of standardized views in all ten.

### PROGRAM: 1_CODE_SYSALLVIEWS3FAM.SAS

**PART 1: Getting a listing of all SYS Views in the ALL_, USER_, and DBA_ families**

```
/*---------------------------------- BEGIN PROGRAM ----------------------------------*/
/* Program Name:  1_Code_SysAllViews3Fam.sas                                        */
/* PART 1: Getting a listing of all SYS Views for ALL_, USER_, and DBA_ families  */

ODS HTML body="&unixpath.&ODBshrt.DB_SysAllViews3Fam.html";
TITLE1 "Breakdown of SYS.ALL_VIEWS in ALL_, USER_, DBA_";
TITLE2 "For ORACLE database &ODBlong.";
PROC SQL;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong" &ODBcred. );
  /* CREATE table SYSallviews as */
  SELECT
  SCANQ(VIEW_NAME,1,"_") as FAMILY,
```

```
  *
  FROM connection to &ODBshrt.
    (
     SELECT
     VIEW_NAME
     FROM SYS.all_views
     WHERE OWNER='SYS'
    )
  WHERE SCANQ(VIEW_NAME,1,"_") in ('ALL','USER','DBA')
  ORDER FAMILY, VIEW_NAME
  ;
  DISCONNECT FROM &ODBshrt.;
  QUIT;
TITLE1;
TITLE2;
ODS HTML close;
/*---------------------------------- END PROGRAM ----------------------------------*/
```

Output 1 provides a snapshot of the results generated from Part 1 of the program for database1. Although there might be some variation to the list from Oracle database to Oracle database, most of the list will be consistent. If you see a particular view of interest to explore further, use Part 2 of the program to look closer at what is contained in the view. I use this approach, along with the online Oracle Database Reference, to expand my knowledge about the Oracle Data Dictionary.

Breakdown of SYS.ALL_VIEWS in ALL_, USER_, DBA_

For ORACLE database database1.server2.com

| FAMILY | VIEW_NAME |
|---|---|
| ALL | ALL_ALL_TABLES |
| ALL | ALL_APPLY |
| ALL | ALL_APPLY_CONFLICT_COLUMNS |
| . | . |
| . | . |
| . | . |
| ALL | ALL_WARNING_SETTINGS |
| DBA | DBA_AUTO_SEGADV_CTL |
| DBA | DBA_AUTO_SEGADV_SUMMARY |
| DBA | DBA_DATA_FILES |
| . | . |
| . | . |
| . | . |
| DBA | DBA_TABLESPACES |
| USER | USER_ADVISOR_ACTIONS |
| USER | USER_ADVISOR_DIRECTIVES |
| USER | USER_ADVISOR_FINDINGS |
| . | . |
| . | . |
| . | . |
| USER | USER_WARNING_SETTINGS |

**Output 1. Sample HTML Output File from PART1 of 1_Code_SysAllViews3Fam.sas.**

## PROGRAM: 1_CODE_SYSALLVIEWS3FAM.SAS

### PART 2: Macro for listing SYS View contents, from a list of view names

Part 2 of the program 1_Code_SysAllViews3Fam.sas is a macro for creating a listing of the first set of observations in each view from a customized list of view names. The output allows you to see both the variable names in the view and what the data elements look like for each variable. This provides a quick way to explore the content of some of the SYS Views.

```
/*---------------------------------- BEGIN PROGRAM ---------------------------------*/
/* Program Name:  1_Code_SysAllViews3Fam.sas                                        */
/* PART 2: Macro for listing SYS View contents, from a list of view names.        */

*Customized the list of SYS Views of interest.  ALL CAPS for view names.;
DATA sysviewlist;
   FORMAT view_name $varying50.;
   INPUT view_name $;
   CARDS;
  USER_ROLE_PRIVS
  ALL_CONS_COLUMNS
  ALL_IND_EXPRESSIONS
  ;
RUN;

*Auto-generate macro variables used to loop the listing.;
DATA _null_;
   SET sysviewlist end=eof;
   CALL SYMPUTX("vname"||CATS(PUT(_N_,3.)), view_name);
   if eof then CALL SYMPUTX("nviews",_N_);
RUN;

*MACRO Definition: list the first set of observations from a SYS View.;
%MACRO prtviews(recs,vname);
TITLE "First &recs rows of view SYS.&vname in &ODBlong.";
PROC SQL inobs=&recs.;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong." &ODBcred. );
  SELECT
  *
  FROM connection to &ODBshrt.
    (
     SELECT
     *
     FROM SYS.&vname.
    )
  ;
  DISCONNECT from &ODBshrt.;
QUIT;
TITLE;
%MEND prntviews;

*Run the loop, once for each view in the list.;
ODS HTML body="&unixpath.&ODBshrt.DB_&nviews.view_listing.html";
%MACRO loop_prntviews;
 %do i=1 %to &nviews;
  %prtviews(recs=15,vname=&&vname&i..); /* can alter number of records here */
 %end;
%MEND loop_prntviews;
%loop_prntviews;
ODS HTML close;
/*----------------------------------- END PROGRAM ----------------------------------*/
```

The HTML file generated in Part 2 of the program 1_Code_SysAllViews3Fam.sas prints the first 15 observations (recs=15) for each of the three views in the customized list in the example (USER_ROLE_PRIVS, ALL_CONS_COLUMNS, ALL_IND_EXPRESSIONS). You can expand the list and modify the number of observations to suit your needs.

## MORE ORACLE® DATA DICTIONARY SYS VIEWS

Although there are hundreds of ALL_ and USER_ SYS Views, we now present the usefulness of a particular handful. In the previous section, we used the view SYS.ALL_VIEWS to explore all the views available. In this section, we will drill down into a short list of views that can be used to answer some key questions about databases.

### USING SYS.ALL_TABLES TO DETERMINE TABLE PARTITIONING

Partitioning enables tables and indexes to be split into smaller, more manageable components and is a key requirement for any large database with high performance and high availability requirements. Due to the nature of partitioning, you should always use a Pass-Through Facility query on a partitioned table. In fact, you might be required to do so depending on how your Database Administrator has things set up.

When I gain access to a new database one of the first things of interest to me is the number of tables in each schema, how large each table is, and which tables are partitioned. All of this can be determined with the use of the ALL_TABLES SYS View. The code presented in this section provides a listing of the table names, table sizes, and a partition flag for a particular schema. The subsequent output reveals the results for a specific set of macro variables.

**Program: 2_Code_SysAllTables.sas**

```
/*---------------------------------- BEGIN PROGRAM ---------------------------------*/
/* Program Name: 2_Code_SysAllTables.sas                                            */
/* Using SYS.ALL_TABLES to Determine Table Partitioning.                            */

ODS HTML body="&unixpath.&ODBshrt.DB_&OWNlong._AllTables.html";
TITLE "Tables and Partitioning on Schema &OWNlong. in Database: &ODBlong.";
PROC SQL;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong." &ODBcred. );
  /* CREATE table alltables as */
  SELECT
  TABLE_NAME, NUM_ROWS FORMAT=comma19.0, PARTITIONED
  FROM connection to &ODBshrt.
    (
     SELECT TABLE_NAME, NUM_ROWS, PARTITIONED
     FROM SYS.all_tables
     WHERE OWNER=&OWNlong. and NUM_ROWS > 0
     ORDER by TABLE_NAME
    )
  ;
  DISCONNECT from &ODBshrt.;
QUIT;
TITLE;
ODS HTML close;


/* Alternate form of the pass-through SQL, to get just partitioned tables:
  (
   SELECT TABLE_NAME, NUM_ROWS, PARTITIONED
   FROM SYS.all_tables
   WHERE OWNER=&OWNlong. and NUM_ROWS > 0 and PARTITIONED='YES'
   ORDER by NUM_ROWS DESC, TABLE_NAME
  )
*/
/*---------------------------------- END PROGRAM ---------------------------------*/
```

Output 2 indicates there are two partitioned tables in the OWNER5 schema of the database database1.server2.com. The largest table has about 233 million records. These two tables will require Pass-Through Facility queries.

Partitioned Tables on Schema 'OWNER5' in Database: database1.server2.com

| TABLE_NAME | NUM_ROWS | PARTITIONED |
|---|---|---|
| TABLE236 | 232,942,060 | YES |
| TABLE285 | 36,673,534 | YES |

**Output 2. Sample HTML Output File from 2_Code_SysAllTables.sas (run with Alternate pass-through SQL)**

## USING SYS.USER_ROLE_PRIVS TO DETERMINE GRANTED DATABASE ROLES AND PRIVILEGES

When you gain access to an Oracle database, you are granted certain roles and privileges. A Role is a way of restricting what objects (schemas, tables, columns) are available to you as a user of the database. Knowing your Role assignments can be critical. If you are getting connection errors in a query it might be because you are not assigned to the proper Role to access the target schema, table, or column. Privileges define the level of access to an object in the database (i.e., Oracle table SELECT, INSERT, DELETE, etc.). Privileges are less important, since as an analyst your privileges will likely be restricted to SELECT only. The code presented in this section provides a listing of the Roles and Privileges assigned to the user account on an Oracle database. The subsequent output reveals the results for a specific set of macro variables.

**Program: 3_Code_SysUserRolePrivs.sas**

```
/*---------------------------------- BEGIN PROGRAM ---------------------------------*/
/* Program Name: 3_Code_SysUserRolePrivs.sas                                        */
/* Using SYS.USER_ROLE_PRIVS to Determine Granted Database Roles and Privileges.  */

ODS HTML body="&unixpath.&ODBshrt.DB_RolesPrivs.html";
TITLE "Roles/Privs granted for &sysuserid. on Database: &ODBlong.";
PROC SQL;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong" &ODBcred. );
  /* CREATE table myroles_&ODBshrt. as*/
  SELECT
  *
  FROM connection to &ODBshrt.
    (
     SELECT *
     FROM SYS.user_role_privs
    )
  ;
  DISCONNECT from &ODBshrt.;
QUIT;
TITLE;
ODS HTML close;
/*---------------------------------- END PROGRAM ---------------------------------*/
```

Output 3 reveals my user account (cjesse) Roles and Privileges for the database database1.server2.com. One of the Roles allows me SELECT access to the OWNER5 schema, and another Role allows me SELECT access to the OWNER8 schema. I could compare my results to those for other analysts on my team to determine transferability of code.

Roles/Privs granted for cjesse on Database: database1.server2.com

| USERNAME | GRANTED_ROLE | ADMIN_OPTION | DEFAULT_ROLE | OS_GRANTED |
|---|---|---|---|---|
| CJESSE | ROLE_OWNER5_READ | NO | YES | NO |
| CJESSE | ROLE_OWNER8_READ | NO | YES | NO |

**Output 3. Sample HTML Output File from 3_Code_SysUserRolePrivs.sas**

## USING SYS.ALL_IND_COLUMNS TO DETERMINE TABLE INDEXES

Just like a SAS data set can be indexed, so can an Oracle table. The purpose of an index is the same in both environments: a performance-tuning method for allowing faster retrieval of records. In other words, indexes are vehicles for speeding up queries. Knowledge about the indexes on an Oracle table can aid in constructing efficient SAS queries involving that table. This becomes increasingly important as the target table gets larger. An Oracle table can have multiple indexes, each based on one or more variables. The code presented in this section provides analysis around the index structure for a particular table in a schema. The subsequent output reveals the results for a specific set of macro variables.

**Program: 4_Code_SysAllIndColumns.sas**

```
/*---------------------------------- BEGIN PROGRAM ---------------------------------*/
/* Program Name: 4_Code_SysAllIndColumns.sas                                        */
/* Using SYS.ALL_IND_COLUMNS to Determine Table Indexes.                            */
```

```
ODS HTML body="&unixpath.&ODBshrt.DB_&OWNlong._&TBL._Indexes.html";
TITLE "Indexes on Schema &OWNlong., Table &TBL. in Database: &ODBlong.";
PROC SQL;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong" &ODBcred. );
  /* CREATE table &TBL.indexes as */
  SELECT
  *
  FROM connection to &ODBshrt.
    (
     SELECT
     INDEX_NAME, COLUMN_POSITION, COLUMN_NAME
     FROM SYS.all_ind_columns
     WHERE TABLE_OWNER=&OWNlong. and TABLE_NAME=&TBL.
     ORDER by TABLE_OWNER, TABLE_NAME, INDEX_NAME, COLUMN_POSITION
    )
  ;
  DISCONNECT from &odbshrt.;
QUIT;
TITLE;
ODS HTML close;
/*---------------------------------- END PROGRAM ----------------------------------*/
```

The results in Output 4 indicate there is only one index on table TABLE11. The index is called TABLE11_PK, involving two variables FIPS_STATE_CODE and FIPS_COUNTY_CODE. If the query sub-set logic on TABLE11 includes the variable FIPS_STATE_CODE, efficiency might be gained by promoting that logic first. Of course, that depends on the size of TABLE11 and the nature of the query. In this case, TABLE11 is not very large, nor is it partitioned. Logic order is not important.

However, consider table TABLE236 in the same schema. It contains hundreds of millions of records. TABLE236 has 15 different indexes on it, ranging from a 1 variable index to an 8 variable index. The results in Output 5, generated for table TABLE236, depict the information for the first 3 indexes (called NI1_TABLE236, NI2_TABLE236, NI3_TABLE236). All three indexes involve BATCH_DATE and ACCOUNT_NUMBER. In fact, BATCH_DATE is the single variable index in the full list of 15. A multi-column subset logic in the pass-through SQL, involving the BATCH_DATE column, will benefit in efficiency by making sure BATCH_DATE is promoted early in the subset logic.

Indexes on Schema 'OWNER5', Table 'TABLE11' in Database: datebase1.server2.com

| INDEX_NAME | COLUMN_POSITION | COLUMN_NAME |
|---|---|---|
| TABLE11_PK | 1 | FIPS_STATE_CODE |
| TABLE11_PK | 2 | FIPS_COUNTY_CODE |

**Output 4. Sample HTML Output File from 4_Code_SysAllIndColumns.sas**

Indexes on Schema 'OWNER5', Table 'TABLE236' in Database: database1.server2.com

| INDEX_NAME | COLUMN_POSITION | COLUMN_NAME |
|---|---|---|
| NI1_TABLE236 | 1 | BATCH_DATE |
| NI1_TABLE236 | 2 | ACCOUNT_NUMBER |
| NI1_TABLE236 | 3 | BAL_PRIN |
| NI1_TABLE236 | 4 | LATE_FEE_UNCOLL |
| NI2_TABLE236 | 1 | BATCH_DATE |
| NI2_TABLE236 | 2 | ACCOUNT_NUMBER |
| NI2_TABLE236 | 3 | STRAT_COLLECTIONS |
| NI2_TABLE236 | 4 | LOAN_STATUS_CODE |
| NI2_TABLE236 | 5 | COLLECTION_RESPONSE_CODE |
| NI2_TABLE236 | 6 | COLLECTOR_ID |
| NI2_TABLE236 | 7 | SUB_SERVICED_IND |
| NI3_TABLE236 | 1 | BATCH_DATE |
| NI3_TABLE236 | 2 | ACCOUNT_NUMBER |
| NI3_TABLE236 | 3 | LOAN_STATUS_CODE |
| NI3_TABLE236 | 4 | CREDIT_MAX |
| NI3_TABLE236 | 5 | BLOCK_NUMBER |
| NI3_TABLE236 | 6 | INVESTOR_NUMBER |
| NI3_TABLE236 | 7 | BAL_PRIN_PRIOR |
| NI3_TABLE236 | 8 | REOPENED_IND |

**Output 5. Sample HTML Output File from 4_Code_SysAllIndColumns.sas**

## USING SYS.ALL_CONSTRAINTS AND SYS.ALL_CONS_COLUMNS TO DETERMINE TABLE PRIMARY KEYS

When querying and merging tables, it is important to understand the granularity of the data; that is, the level to which rows are uniquely identified within the table. The importance comes in understanding when you might be merging one-to-one, one-to-many, and many-to-many. Primary Keys in Oracle are useful for this purpose. A Primary Key is a single variable or combination of variables that uniquely identifies a record. A table can have only one Primary Key. The two SYS Views that contain Primary Key information are ALL_CONSTRAINTS and ALL_CONS_COLUMNS. The code presented in this section provides listings of the Primary Key structure for an entire database, and more specifically for a particular table in a schema. The subsequent figure reveals the output for a specific set of macro variables.

**Program: 5_Code_Sys_PrimaryKeys.sas**

```
/*--------------------------------- BEGIN PROGRAM ---------------------------------*/
/* Program Name: 5_Code_Sys_PrimaryKeys.sas                                        */
/* Using SYS ALL_CONSTRAINTS & ALL_CONS_COLUMNS to Determine Table Primary Keys.  */

/* ALL Primary Keys in the DB */
ODS HTML body="&unixpath.&ODBshrt.DB_PrimaryKeys.html";
TITLE "All Primary Keys in the ORACLE database: &ODBlong.";
PROC SQL;
  CONNECT to oracle as &ODBshrt. (path="&ODBlong." &ODBcred. );
  /*CREATE table allPriKeys as*/
  SELECT
  *
  FROM connection to &ODBshrt.
    (
     SELECT
     b.OWNER,
     a.TABLE_NAME,
     a.COLUMN_NAME,
     a.POSITION,
     b.STATUS
```

```
    FROM SYS.all_constraints b, SYS.all_cons_columns a
    WHERE
        b.CONSTRAINT_TYPE = 'P'
    AND b.CONSTRAINT_NAME = a.CONSTRAINT_NAME
    AND b.OWNER = a.OWNER
    ORDER BY b.OWNER, a.TABLE_NAME, a.POSITION
   )
  ;
  DISCONNECT from &ODBshrt.;
QUIT;
TITLE;
ODS HTML close;

/* Alternate form of the pass-through SQL, to get a Primary Keys on specific table.
  (
   SELECT
   b.OWNER,
   a.TABLE_NAME,
   a.COLUMN_NAME,
   a.POSITION,
   b.STATUS
   FROM SYS.all_constraints b, SYS.all_cons_columns a
   WHERE
       a.TABLE_NAME = &TBL.
   AND b.CONSTRAINT_TYPE = 'P'
   AND b.CONSTRAINT_NAME = a.CONSTRAINT_NAME
   AND b.OWNER = a.OWNER
   ORDER BY b.OWNER, a.TABLE_NAME, a.POSITION
  )
*/
/*---------------------------------- END PROGRAM ----------------------------------*/
```

Output 6 reveals the results for the table called TABLE1 in the OWNER1 schema of the database database1.server1.com. The variables ASSET_SEQ_ID and PAYOFF_TYPE_CODE are the likely candidates for merging this table.

All Primary Keys in Table 'TABLE1' in the ORACLE database: database1.server1.com

| OWNER | TABLE_NAME | COLUMN_NAME | POSITION | STATUS |
|---|---|---|---|---|
| OWNER1 | TABLE1 | ASSET_SEQ_ID | 1 | ENABLED |
| OWNER1 | TABLE1 | PAYOFF_TYPE_CODE | 2 | ENABLED |

**Output 6. Sample HTML Output File from 5_Code_Sys_PrimaryKeys.sas (with Alternate pass-through SQL)**

## CONCLUSION

This paper began with an overview of Oracle database architecture and review of PROC SQL query syntax with SAS/ACCESS Interface to ORACLE. I have presented a handful of simple programs to get you started on your own exploration of the SYS Views, the backbone to Oracle database metadata. These programs are building blocks to understanding the SYS Views and can aid you in understanding your Oracle data sources. Utilizing SAS to explore the Oracle Data Dictionary is the Getting-to-Know-You Phase of Romancing Your Data.

## REFERENCES

Oracle. "The Oracle® Database Reference, 10g Release 2 (10.2)." Available at http://download.oracle.com/docs/cd/B19306_01/server.102/b14237/toc.htm

SAS Institute Inc. "SAS OnlineDoc® documentation, SAS/ACCESS® 9.1.3 for Relational Databases: Reference; SAS/ACCESS for ORACLE." Copyright 2007, SAS Institute Inc., Cary, NC, USA. All Rights Reserved. Reproduced with permission of SAS Institute Inc., Cary, NC. Available at http://support.sas.com/onlinedoc/913/getDoc/en/acreldb.hlp/a001386257.htm

## ACKNOWLEDGEMENTS

I extend my thanks to two of my Twitter cohorts- first, to Gordon T. Cox (on Twitter as @gtcox76), Technology Applications Consultant at Humana, who inspired me to learn more about the Oracle Data Dictionary Views. Gordon also provided invaluable technical review comments on this paper. Second, I am indebted to Vicki Boykis (on Twitter as @vboykis), Economic Consultant and Technical Writer, for her editorial help with the document review. The value I place on having access to an actual Writer and Technical Editor for the review of this paper is immeasurable. Vicki is my go-to Writer person. This paper is a testament to the power of Twitter for fostering professional relationships in a virtual world.

I would also like to thank Sue Douglass, the Data Integration Section Chair, for accepting this abstract and paper to the SAS Global Forum 2011.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author: Carole Jesse
Email: Carole.Jesse@gmail.com
LinkedIn: http://www.linkedin.com/in/carolejesse
Twitter: http://www.twitter.com/CaroleJesse
SASCommunity: http://www.sascommunity.org/wiki/User:CaroleJesse